Object Oriented Principles in Software

Hassan Rashidi[1]

Abstract

One of the modern approaches to develop a system is object oriented analysis and design.

In this approach, there are several objects and each object plays some specific roles. These roles are programmed in object oriented languages such C++ and Java. This paper describes eight basic principles and key concepts in Object Oriented technology. These principles are Encapsulation, Information Hiding, Message Passing, Late Binding, Delegation, Instantiation, Inheritance and Polymorphism, and Relationships.

Keywords: Class, Object, Object-Oriented, Software Engineering

1. Department of Mathematics, and Computer Science, Allameh Tabataba'i University, Tehran, Iran Email: hrashi@gmail.com;hrashi@atu.ac.ir

1-Introduction

Object-oriented technology defines a set of theories, standards, and methods that together represent a way of organizing knowledge. This technology is a very similar to a style of collaborating objects. A program is a lattice of objects connected together according to identified relationships. The computing/Operations is achieved by passing messages among objects, delegating responsibilities for services to other objects, and providing services by exhibiting the appropriate behavior for a given message.

Maintainable, flexible, and reliable software is difficult to produce. Software systems are complex and, as suggested by Brooks, complexity is a part of the essence of the system. No process of abstraction can eliminate complexity in its entirety. However, we believe that we can create mechanisms that help us manage these complexities in the object oriented technology.

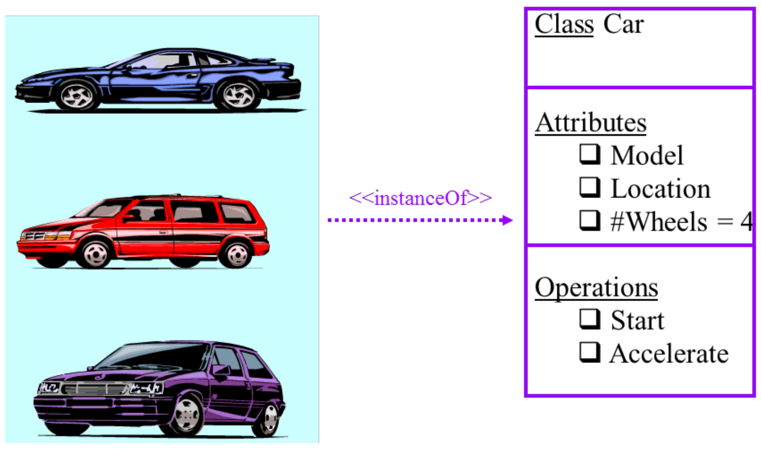
The structure of remaining sections of this paper is as follows. In Section 2, we introduce object-oriented concepts. In Section 3, the object-oriented principles are presented. In Section 4, object-oriented programming languages are classified. Section 5 is considered to summary and conclusion.

2-Object-Oriented Concepts

The most fundamental concept and mechanism of object-oriented programming is an *object*. We define an object as a software unit consisting of the attributes (data) and the methods (code) that act on those data. The data is not directly accessible to the users of the object. Access to the data is granted only via the methods, or code, provided by object (i.e., function calls to its methods).

A class is group of entities/objects that have the same attributes and specific behaviors or operations as services. As shown in Fig.1, suppose Class Car with three attributes and a couple of services. All objects are instances of the classes. For example, three specific cars are instances of Class car.

Note that the instances can be created or destroyed at run time and each Object has to provide at least one Service or Behavior for other objects.
In the object-oriented modeling/ programming, we define various objects in the universe that will help us solve the problem as well as how they interact with each other, and then we set them in motion. As a result, we consider object-oriented programming as using a simulation model of computation instead of a pigeon-hole model of computation. This simulation model also provides the designer / programmer with a better metaphor for problem solving. When we think in terms of services and methods (how to provide the service), we can bring a wealth of experience, understanding, ideas, and intuition from our everyday lives. In contrast, most of us have very little insight into how to structure a program that thinks about problem solving in terms of pigeon holes or slots containing values.



.Figure 1: Class (right) and three instances of class car (left) as Objects

3-Object-Oriented Principles

This section defines the basic principle of the object-oriented paradigm.

3-1- Encapsulation

As shown in Figure 2, the object contains both the data and the methods (code) that will manipulate or change that data. The services of an object define how other objects gain access to its methods. Each object advertises the public services it is willing to provide to all objects. It also provides other services (protected-and private) that are restricted only to specific other objects.

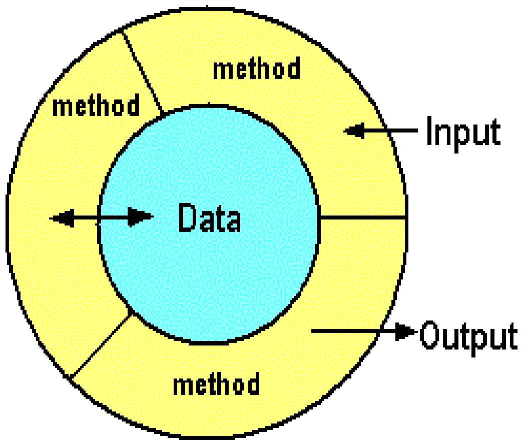


Figure 2: Object has data encapsulated by functions.

3-2- Information Hiding

The idea of providing services defines the second principle of the object-oriented paradigm-information hiding. The object that contains the attributes (data) defines what services (functions) are available to the other objects. In fact, other objects have neither access nor knowledge of the data (attributes) or how (method/ code) a service is provided.

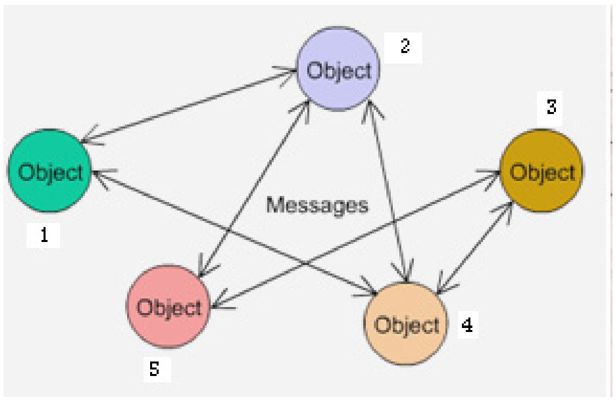
Let us define an agent as an object that provides services and a client as an object that uses one or more of these services. An agent may also be called a server. An agent may be better term during analysis, as a server may then be reserved to mean the actual object that does the work. Before a client may use an agent's services, an interface must be defined. This interface definition is called the prototype of the service. The prototype is made from two parts: (a) the name of the service (called the selector by some experts), and (b) the arguments for the service- (called the signature by some experts).Every object must define its prototype for each service it plans to provide. The set of defined prototypes is the protocol of the object (or, alternatively, it is the object's interface).The protocol defines how a client may invoke (or request) the services of this object.

A good example of an object and its interface is the icon system that we all use in a wind owing system. A very common action in this system is to select an icon and then use a pull-down menu to get all of the services that we can choose for that icon. Thus, in an object-oriented system, the icon is really an object and the menu defines the object interface (or protocol).

One object can use the public service of another object by using the message-passing mechanism (paradigm) to send a message that conforms to the prototype of the service. If object 1 (the client) wants to use a service of object 2 (the agent), the client sends a message to the agent requesting the specific service of the agent. Note that the message must be directed to a specific object and contain the name- of the requested service. Furthermore, the message may contain additional information (arguments) needed by the agent to perform the requested service.

3-3- Message Passing

In the object oriented approach, an object may communicate with another object only via the message-passing mechanism. Each message must be sent to a designated receiver, and the interpretation of the message depends on the receiver. In the object-oriented paradigm, the specific receiver of any given message is not usually known until run time, so the determination of which method to invoke cannot be made until then. Thus, there is late binding between the message (service request/function call) and the method (code fragment) that will be used to fulfill the request for action. This can be contrasted to the early binding (compile or link time) of the function call to the code fragment in the imperative-programming paradigm. The support for late binding defines the fourth principle of the object-oriented paradigm-late binding.



.**Figure 3:** Message passing among objects

For example, in the programmer's parlance, object 1 makes a function call to the service (function) that belongs to object 2 and passes all the appropriate parameter values needed by the function call. Since we stated that message passing is implemented by a function call in C++, it is fair to ask in what sense a message-passing mechanism is different from a function-call mechanism. Certainly, in both cases there is an implicit request for action and there is a set of well-defined operations that will be performed to fulfill the request. However, there are three important distinctions. First, in a message-passing mechanism, each message is sent to a designated receiver (agent). In the imperative-programming paradigm, a function-call mechanism has no designated receiver (agent). This distinction supports encapsulation. Second, the interpretation of the message (method or set of operations/ code used to fulfill the service request) depends on the receiver and can vary with different receivers. This distinction is necessary to support information hiding and polymorphism, which we will explain later.

3-4- Late Binding

Support for the ability to determine the specific receiver and its corresponding method (code) to be executed for a message at run time. ‘From a client’s perspective, it is the agent that provides the service. It is possible that the agent actually delegates the work to a third object. This leads to the fourth principle of the object-oriented paradigm-delegation.

3-5- Delegation

In the object oriented approach, work is passed, via message passing, from one object (client) to another object (agent) because, from the client’s perspective, the agent has the services that the client needs. Work is continuously passed until it reaches the object that has both the data and the method (code) to perform the work.

Delegation is sometimes referred to as the perfect bureaucratic principle. Consider, for example, a corporation or a governmental organization. The chairperson of the board sends a service request (message) to the chief operating officer to build a new car-park in Allameh University. From the perspective of the chairperson of the board, it is the chief operating officer’s responsibility to provide the service. However, we all know that the chief operating officer has neither the skills nor the knowledge (information) to actually build a car-park in Allameh University. So the chief operating officer has a method that delegates the work to the head of projects. The head of projects has a method that delegates the work to the chief engineer who has the staff to build a car-park in Allameh University. In fact, the chief engineer’s method delegates specific tasks to the appropriate heads of various disciplines to build the car-park. It is the specific engineers who have the knowledge and the skills to design the car-park to be built.

Here we see the fifth principle of the object-oriented paradigm applied. The work is delegated to the object that has the information (data) and the skills (method) to perform the task. The bureaucratic part comes from the fact that both the chief operating officer and the head of projects advertise the service "build a new car-park," even though neither one has the information or the skills to perform the task. However, they have access to resources (objects) that can perform the task, so they can then take the responsibility for performing the task.

What got delegated is the authority to get the work done; responsibility cannot be delegated. From the chairperson of the board's perspective, it is the responsibility of the chief operating officer to fulfill the request to build a new car-park in Allameh University: Similarly, from the chief operating officer's perspective, it is the responsibility of the head of projects to fulfill the request to build a new car-park in Allameh University. When the head of projects accepts this request, he/ she has accepted the responsibility from the chief operating officer to perform the work. However, neither the chief operating officer nor the chairperson knows how the work will be done. This is applying the information-hiding principle.

3-6- Instantiation

Now let us look at some other object-oriented -concepts, Categorizing helps us organize the complex world in which we live. We can make certain assumptions about an object that is in a particular category. If an object is an instance of the category (class),it will fit the general pattern for that category. This leads us to the sixth principle of the object-oriented paradigm-class / instance / object.

All objects are instances of a class. Instances can be created (instantiated) or destroyed (deleted) at run time. How the object provides a service is determined by the class of which the object is an instance. Thus, all objects of the same class use the same method (code) in response to a specific service request (function call). Earlier we discussed the prototype of a service and the protocol as it relates to an object. Now with the concept of a class, we see that the prototypes and the protocol are really defined for a class and that they are applicable to every object that is an instance of that class.

3-7- Inheritance and Polymorphism

In the object oriented approach, we do not only organize our objects into categories (classes), but we also arrange our categories into a hierarchy from the general to the specific. This leads us to the seventh principle of the object-oriented paradigm-generalization (without/with polymorphism).

- **Generalization Without Polymorphism**. Classes can be organized by using a hierarchical inheritance structure. In the structure, the subclass will inherit the attributes, the relationships, and the methods from the superclass that are higher in the tree. An abstract superclass is a class that is used to create only subclasses. Thus, there are no direct instances of that class. However, there are always exceptions to the rule. In order to handle exceptions to the rule within our hierarchal structure, we define the seventh principle of the object-oriented paradigm-generalization with polymorphism, a modification of the sixth principle.
- **Generalization With Polymorphism**. Classes can be organized by using a hierarchal inheritance structure. In the structure, the subclass will inherit the attributes, relationships, and methods from the superclass that is higher in the tree. However, a subclass may create its own method to replace a method of any of its superclasses in providing a service that is available at the superclass level when an instance of that subclass is the agent. For the subclass, its method will *override* the superclass method for providing the same service. Although generalization is a powerful concept, there are relationships between objects that cannot be captured using this concept. For suppose different cars as shown in Fig. 4. Some cars in this hierarchy have the same operations that are defined in the class above.

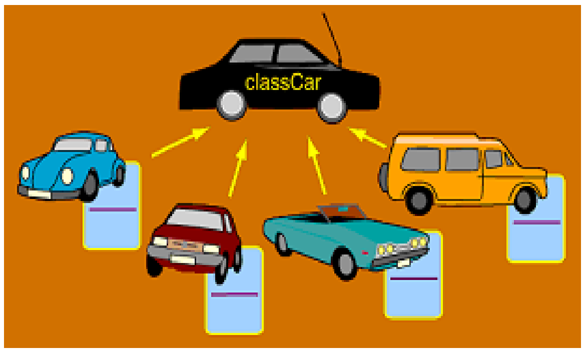


Figure 4: An example of inheritance and polymorphism

3-8- Relationships

In the object oriented software, collaborations between objects to provide a service to a client are usually captured by an association relationship, which is technically called a link. Moreover, there is another relation among objects, known as aggregation relationship. In this relationship, several objects are combined to define an aggregated object.

4-Object-Oriented Programming Languages

The problem-solving view of object-oriented programming is very different from the pigeon-hole model used in imperative programming. In the object-oriented paradigm, we never used any of the conventional terms such as assignments, variables, or memory addresses. Instead, we use terms such as objects, messages, and services. The application frameworks usually have a universe of well-behaved objects that courteously ask each other to perform services for themselves.

In the managing of object oriented software development, The idea of creating a universe of helpers is very similar to a style of computer simulation called" discrete event-driven simulation." In this style, the user creates models of various elements of the simulation and describes how they interact with each other. Then, via some discrete event, the elements are set in motion.

Object-oriented programming is an advanced programming paradigm. Before going to the implementation, it is suggested to look briefly at the programming concepts and the variety of languages used in object-oriented technology. The programming languages are described in the following.

4-1- Object-based Programming

Object-based programming is a programming style that uses encapsulation and objects. The methods and attributes are hidden within the object, and each object is uniquely identified. There is no support for the class mechanism, inheritance, relationships, dynamic behavior, and rules. Ada is an example of an object-based language. Visual Basic is also an example of an object-based programming language

4-2- Class-based Programming

Class-based programming includes all of the mechanisms of object-based languages as well as the mechanisms for classes and instances. CLU is an example of a class-based programming language.

4-3- Object-oriented Programming

Object-oriented programming includes all of the mechanisms for class-based programming as well as mechanisms to support inheritance and self-recursion. Smalltalk is an example of an object-oriented programming language.

4-4- Advanced OO Programming

Advanced OO Programming includes all of the mechanisms for object-oriented programming and includes capabilities to support multiple inheritance, association, aggregation, and dynamic behavior. C++ and Java are examples of advanced OO programming languages.

4-5-Leading-edge Object-Oriented Programming

Leading-edge OO programming includes all of the mechanisms of an advanced OO programming style as well as mechanisms for supporting implementation of rules. R++ is an example of such a leading-edge programming language.

5-Summary and Conclusion

The principles of the object-oriented paradigm are as follows: (a) **Encapsulation**: An object contains both the data and the methods (code) that will manipulate or change the data; (b)**Information hiding:** The services of an object define how other objects have access to its methods and, therefore, its data. Each object advertises public services that it is willing to provide to other objects; (c) **Message passing**: An object (client) may communicate with another object (agent) only via the message-passing mechanism. A client requests a service of an agent by sending a message that matches a predefined protocol that the agent defines for that service; (d) **Late binding:** The specific receiver of any given message will not be known until run time, so the determination of which method to invoke cannot be made until then. Remember inheritance with polymorphism; (e) **Delegation:** Work is passed, via the message-passing mechanism, from one object (client) to another object (agent) because from the client's perspective the agent has the services that the client needs. Work is continuously passed until it reaches the object that has both the data and the method (code) to perform the work. Delegation is sometimes called the perfect bureaucratic principle; (f) **Class and objects**. All objects are instances of a class. How an object provides a service is determined by the class of which the object is an instance. Thus, all objects of the same class use the same method (code) in response to a specific service request; (g) **Inheritance and polymorphism:** Classes can be organized by using a hierarchal inheritance structure. In the structure, the subclass will inherit the attributes and the methods from the superclass (es) higher in the tree. However, a subclass may create its own method to replace the method of any of its superclasses in providing a service that is available at the superclass level. When the subclass is the agent for that specific service, the method of the subclass will override the method of the superclass (es) for providing the same service; (h) **Relationships**: Association and aggregation are used to capture the collaboration between objects necessary to provide a service to a client. In the object oriented software development, we must consider that objects are "black boxes" which send and receive messages. Note that object Oriented approach requires a major shift in thinking by designers/programmers.

We believe that object-oriented offers a new and powerful model for developing systems/software. This approach speeds the development of new programs, and, if properly used, improves the maintenance, reusability, and modifiability of software. Today, Agents (as the matured Objects) are developed and used to operate in business, economy, education, manufacturing, game and even research!.

References

Lee C. and Tepfenhart M., “UML and C++: A Practical Guide To Object-Oriented Development”, Pearson Prentice Hall, 2005.

Bruegge B. and Dutoit A.H., “Object-Oriented Software Engineering”, 2nd Edition, Pearson Prentice Hall, 2010

Pressman R.S. and Ince D. , “Software Engineering (European Edition)”, McGraw Hill, 2012 .

Sommervile I., “Software Engineering”, McGraw Hill, 2016

Pfleeger S.L., “Software Engineering - Theory and Practice”, Prentice Hall, 2010.

Rashidi H., “Software Engineering- A programming Approach”, Allameh Tabataba’i Press (in Persian), 2015